// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract AssetToken is ERC20 {

uint256 public assetPerToken = 1; // number of assets created per token deposit

uint256 public rewardPerAsset = 0.1 ether; // amount of reward per asset per day

uint256 public rewardPool = 10000 ether; // total reward pool size

struct Asset {

uint256 createdAt;

uint256 lastClaimedAt;

}

mapping(address => Asset[]) public assets;

event Deposit(address indexed from, uint256 value);

event AssetCreated(address indexed from, uint256 count);

event RewardClaimed(address indexed from, uint256 count);

constructor() ERC20("Asset Token", "ASSET") {}

function deposit(uint256 value) external {

require(value % 10 == 0, "Value must be multiple of 10");

\_mint(msg.sender, value);

emit Deposit(msg.sender, value);

uint256 assetCount = value / 10;

for (uint256 i = 0; i < assetCount; i++) {

assets[msg.sender].push(Asset(block.timestamp, block.timestamp));

}

emit AssetCreated(msg.sender, assetCount);

}

function claimRewards(uint256 startIndex, uint256 endIndex) external {

uint256 totalReward = 0;

require(startIndex < endIndex, "Invalid index range");

for (uint256 i = startIndex; i < endIndex; i++) {

Asset storage asset = assets[msg.sender][i];

require(asset.createdAt > 0, "Asset does not exist");

uint256 rewardableDays = (block.timestamp - asset.lastClaimedAt) / 1 days;

if (rewardableDays > 0) {

uint256 reward = rewardableDays \* rewardPerAsset;

if (reward > rewardPool) {

reward = rewardPool;

rewardPool = 0;

} else {

rewardPool -= reward;

}

totalReward += reward;

asset.lastClaimedAt += rewardableDays \* 1 days;

}

}

require(totalReward > 0, "No rewards to claim");

\_mint(msg.sender, totalReward);

emit RewardClaimed(msg.sender, totalReward);

}

}

**ASSIGNMENT 2**

small visual tester written in JavaScript using React framework to test the functionality of the AssetToken smart contract:

import React, { useState, useEffect } from "react";

import Web3 from "web3";

import AssetToken from "./contracts/AssetToken.json";

const web3 = new Web3(Web3.givenProvider || "http://localhost:8545");

const contractAddress = "0x0123456789abcdef0123456789abcdef0123456";

function App() {

const [account, setAccount] = useState(null);

const [tokenBalance, setTokenBalance] = useState(0);

const [assetCount, setAssetCount] = useState(0);

const [rewardBalance, setRewardBalance] = useState(0);

const [loading, setLoading] = useState(false);

const [error, setError] = useState(null);

useEffect(() => {

const loadAccount = async () => {

const accounts = await web3.eth.getAccounts();

setAccount(accounts[0]);

};

const loadTokenBalance = async () => {

const contract = new web3.eth.Contract(AssetToken.abi, contractAddress);

const balance = await contract.methods.balanceOf(account).call();

setTokenBalance(balance);

};

const loadAssetCount = async () => {

const contract = new web3.eth.Contract(AssetToken.abi, contractAddress);

const assetCount = await contract.methods.assets(account).call();

setAssetCount(assetCount);

};

const loadRewardBalance = async () => {

const contract = new web3.eth.Contract(AssetToken.abi, contractAddress);

const rewardBalance = await contract.methods.rewardPool().call();

setRewardBalance(rewardBalance);

};

setLoading(true);

setError(null);

loadAccount()

.then(() => loadTokenBalance())

.then(() => loadAssetCount())

.then(() => loadRewardBalance())

.catch((error) => setError(error.message))

.finally(() => setLoading(false));

}, [account]);

const handleDeposit = async (amount) => {

setLoading(true);

setError(null);

const contract = new web3.eth.Contract(AssetToken.abi, contractAddress);

await contract.methods.deposit(amount).send({ from: account });

setLoading(false);

window.location.reload();

};

const handleClaimRewards = async () => {

setLoading(true);

setError(null);

const contract = new web3.eth.Contract(AssetToken.abi, contractAddress);

const assetCount = await contract.methods.assets(account).call();

const startIndex = 0;

const endIndex = assetCount;

await contract.methods.claimRewards(startIndex, endIndex).send({ from: account });

setLoading(false);

window.location.reload();

};

return (

<div>

{loading ? <p>Loading...</p> : null}

{error ? <p>Error: {error}</p> : null}

{account ? <p>Connected account: {account}</p> : null}

<p>Token balance: {tokenBalance}</p>

<p>Asset count: {assetCount}</p>

<p>Reward balance: {rewardBalance}</p>

<button onClick={() => handleDeposit(10)}>Deposit 10 tokens</button>

<button onClick={() => handleDeposit(20)}>Deposit 20 tokens</button>

<button onClick={() => handleClaimRewards()}>Claim rewards</button>

</div>

);

}

export default App;